Programación Python para Machine Learning

Lección 6: Regresión Lineal, Regresión Logística y kNN.

eip ESCUELA INTERNACIONAL POSGRADOS

ÍNDICE

Lección 6: Regresión Lineal, Regresión Logística y kNN.

1. Introducción

En esta lección se estudiarán tres de las técnicas básicas supervisadas dentro del Machine Learning. Pese a su simpleza, son modelos que son realmente útiles en un gran número de casos. De este modo, se plantearán los principios teóricos y fundamentos de la regresión lineal, tanto simple como múltiple, así como el modo de implementarlos en Python para resolver problemas de regresión.

Objetivos

- Comprender los conceptos fundamentales de la Regresión Lineal simple y múltiple.
- Conocer las herramientas para implementar una Regresión Lineal para problemas de regresión.
- Exponer los conceptos fundamentales de la Regresión Logística simple y múltiple.
- Implementar una Regresión Logística para resolver problemas de clasificación binarios y multiclase.
- Comprender los conceptos en los que se basa el modelo k-Vecinos más cercanos (kNN).
- Saber implementar un modelo kNN para problemas de clasificación y regresión.

Por otro lado, se mostrarán las bases teóricas de la regresión logística simple y múltiple, y cómo implementar en Python esta técnica para resolver problemas de clasificación binarios y multiclase.

Finalmente, se expondrán las bases de los modelos k-Vecinos más cercanos (k-Nearest Neighbours, kNN), y cómo implementar en Python esta técnica para resolver problemas de clasificación binarios y multiclase y problemas de regresión.

2. Regresión Lineal

La Regresión Lineal simple es un procedimiento que permite encontrar una relación lineal entre dos variables mediante el cálculo de una línea recta.

En la Ilustración 1 se puede ver un ejemplo gráfico del resultado del proceso. La relación lineal entre dos variables se traduce en que un incremento en la variable independiente genera un incremento proporcional en la variable dependiente.

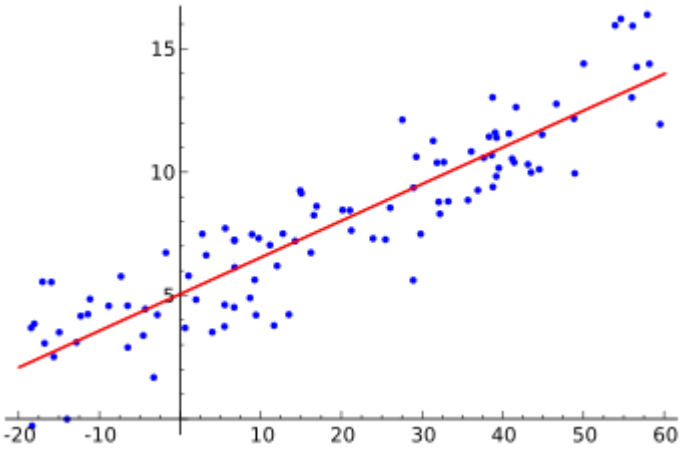


Ilustración 1: Ejemplo de regresión lineal simple.

Se parte de la ecuación de una línea recta:

$$y = wx + b$$

En la que, *y* es la variable dependiente, *x* la variable independiente, *w* la pendiente de la recta y *b* la intersección con el eje de ordenadas. El método de regresión lineal calcula el valor óptimo para la intersección y la pendiente a partir de una serie de puntos (*x,y*).

La regresión lineal simple no tiene porqué buscar valores mediante un cálculo analítico. Es posible estimar *w* y *b* de modo directo utilizando los datos disponibles:

$$w = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} = \frac{Cov(x,y)}{Var(x)}; \qquad b = \bar{y} - w\bar{x};$$

Siendo $\bar{x}$ la media muestral de la variable independiente, $\bar{y}$ la media de la dependiente, $Cov(x, y)$ la covarianza de las dos variables y $Var(x)$la varianza de la variable independiente.

El mismo concepto se puede ser extendido a la situación en los que existan más de dos dimensiones. Es la denominada regresión lineal múltiple. En tal caso, la variable dependiente se explica a través de distintas variables independientes y la separación vendrá dada por un hiperplano:

$$y = w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + w_0$$

Para calcular los coeficientes w_i, se puede plantear el problema de modo matricial y resolverlo por el método de mínimos cuadrados ordinario.

Un modelo de regresión lineal por mínimos cuadrados es una técnica paramétrica, puesto que asume una serie de condiciones sobre las que basa su desarrollo matemático. En los datos de problemas del mundo real, rara vez se cumplen tales asunciones. Sin embargo, esto no se traduce en que el modelo no sea útil. De manera práctica, se trata de ser tener consciencia de ellas y de la repercusión que tiene este hecho en las conclusiones que son susceptibles de ser extraídas del modelo.

Las principales condiciones que asume la regresión lineal son:

- Independencia entre las variables entrada: Las variables de entrada deben ser independientes, no debe de haber colinealidad entre ellas. La colinealidad es un término para referirse a la situación en la que una variable de entrada está linealmente relacionada con una o varias de las otras variables de entrada del modelo.
- Distribución normal de la variable de salida.
- No autocorrelación: Los valores de cada instancia para cada variable son independientes del valor de las variables de las demás instancias. Especialmente, esto se hace importante cuando se trabaja con datos temporales.

3. Implementación en Python de una Regresión Lineal

Por sus características, los modelos de Regresión Lineal, tanto simple como múltiple, son una técnica apropiada para resolver problemas de regresión. En Python es posible implementar este tipo de algoritmo utilizando la clase `LinearRegression` dentro del módulo `linear_model` de scikit-learn.

Para ilustrar el siguiente ejemplo se hará uso del conjunto de datos *housing* del repositorio *UCI Machine Learning.*

En todo caso, hay que recordar que cuando se trata con un modelo en scikit-learn, todas las variables deben ser de numéricas. Por tanto, toda variable categórica en el conjunto de datos debe ser convertidas en numéricas del modo más adecuado y representativo posible.

```
import numpy, random, time
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from pandas import read_csv
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import RobustScaler,MinMaxScaler,
StandardScaler

seed=random.seed(time.time())
filename = '../datasets/housing.csv'

col_names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS',
'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']
data = read_csv(filename, names=col_names, sep=" ")

X = data[data.columns[:-1]]
Y = data['MEDV']

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size =
0.3, random_state=seed)

stdScaler = MinMaxScaler().fit(X_train)
X_train = stdScaler.transform(X_train)
X_test = stdScaler.transform(X_test)

modelo = LinearRegression()
modelo.fit(X_train, y_train)
y_pred = modelo.predict(X_test)

print(modelo.coef_)
rmse = (numpy.sqrt(mean_squared_error(y_test, y_pred)))
print(rmse)
```

4. Regresión Logística

La Regresión Logística es otro ejemplo de modelo supervisado lineal de Machine Learning heredado de la estadística. Al igual que ocurre con la Regresión Lineal, este tipo de método representa uno de los métodos básicos de los que partir para resolver problemas de clasificación, aunque su nombre mencione la palabra regresión. El término logística deriva de la función en la que se fundamenta, la función logística, una curva en forma de *'S'* que proyecta valores reales en el rango entero (0, 1), como puede observarse en la Ilustración 2.

$$Sigmoide(x) = \frac{1}{1 + e^{-x}}$$

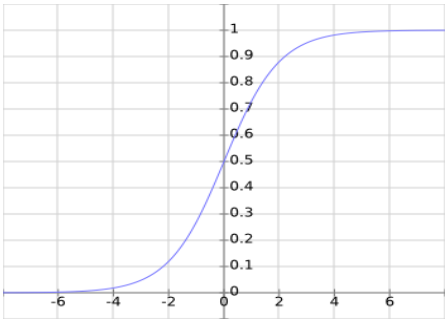


Ilustración 2: Función de activación Sigmoide.

El método en sí es un intento de modelar mediante una o más variables independientes la probabilidad de una variable dependiente binaria. Cuando solo hay una variable independiente se denomina regresión logística simple. En el caso de que haya más de una, regresión logística múltiple.

Si se sustituye y de la función sigmoide por la función de un modelo lineal, $y = w_1x_1 + w_2x_2 + \ldots + w_nx_n + w_0$, se obtiene que:

$$P(y = 1 | X = x) = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 \ldots + w_nx_n + w_0)}}$$

De esta manera se podría calcular la probabilidad de que un patrón pertenezca a la clase 1 sabiendo las variables de entrada x. Aunque directamente desde esta ecuación no pueden calcularse los w_i, mediante una serie de transformaciones se podría tener la posibilidad de aplicar el método de los mínimos cuadrados ordinarios para obtener los coeficientes.

Al igual que los modelos de regresión lineal, la regresión logística es una técnica paramétrica, que asume del mismo modo una serie de condiciones sobre las que basa su desarrollo. En los datos de problemas del mundo real, rara vez se cumplen tales asunciones. Esto no se traduce en que el modelo no sea útil, si no que, de manera práctica, se trata de ser tener consciencia de ellas y de la repercusión que tiene este hecho en las conclusiones que son susceptibles de ser extraídas del modelo.

Las principales condiciones que asume la regresión logística son:

- Independencia entre las variables entrada: Las variables de entrada deben ser independientes entre sí, no debe de haber colinealidad entre ellas. Es decir, si una variable de entrada está linealmente relacionada con una o varias de las otras variables de entrada del modelo, el rendimiento del modelo se verá afectado negativamente.
- Distribución normal de la variable de salida.
- No autocorrelación: Los valores de cada instancia para cada variable deben ser independientes del valor de las variables de las demás instancias. Especialmente, esto se hace importante cuando se trabaja con datos temporales.

5. Implementación en Python de una Regresión Logística

Por sus características, los modelos de Regresión Logística, tanto simple como múltiple, son una técnica apropiada para resolver problemas de clasificación binaria o multiclase. En Python es posible implementar este tipo de algoritmo utilizando la clase `LogisticRegression` dentro del módulo `linear_model` de scikit-learn.

Para ilustrar el siguiente ejemplo se hará uso del conjunto de datos *magic del* repositorio *UCI Machine Learning*.

En todo caso, hay que recordar que cuando se trata con un modelo en scikit-learn, todas las variables deben ser de numéricas. Por tanto, toda variable categórica en el conjunto de datos debe ser convertidas en numéricas del modo más adecuado y representativo posible.

```
import random, time
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from pandas import read_csv
from sklearn.metrics import confusion_matrix, accuracy_score,
balanced_accuracy_score
from sklearn.preprocessing import StandardScaler

seed=random.seed(time.time())

filename = '../datasets/magic04.data'
col_names=['fLength','fWidth','fSize','fConc','fConc1','fAsym','fM3Lon
g','fM3Trans','fAlpha','fDist','class']
data = read_csv(filename, names=col_names)

X = data[data.columns[:-1]]
Y = data['class']

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size =
0.5, random_state=seed)

stdScaler = StandardScaler().fit(X_train)
X_train = stdScaler.transform(X_train)

modelo = LogisticRegression()
modelo.fit(X_train, y_train)

X_test = stdScaler.transform(X_test)
y_pred = modelo.predict(X_test)


cm = confusion_matrix(y_test, y_pred)
```

```
acc= accuracy_score(y_test, y_pred)
bacc = balanced_accuracy_score(y_test, y_pred)
print(cm)
print(acc)
print(bacc)
```

5.1. Clasificación multiclase

En su naturaleza, la regresión logística resuelve problemas de clasificación binarios. Sin embargo, existen estrategias para poder escalar sus principios a problemas en los que los patrones pertenecen a más de dos clases.

- La estrategia Uno-contra-Todos (OVR): divide una clasificación multiclase en un problema de clasificación binaria de los patrones de cada clase contra los patrones del resto de clases en conjunto. Si hay *n* clases, se generan *n* modelos. La predicción del patrón será en base a la máxima salida entre todos los modelos.
- La estrategia Uno-contra-Uno (OVO): divide una clasificación multiclase en un problema de clasificación binaria por cada par de clases, generándose n(n-1)/2 modelos. La predicción del patrón será en base un proceso de votación de todos los modelos.

Por defecto `LogisticRegression` de scikit-learn trae por defecto configurada la estrategia OVR, siendo totalmente transparente al programador esta característica. En caso de que se necesite la utilización de la estrategia OVO, se puede configurar mediante el parámetro `multi_class= 'ovo'`.

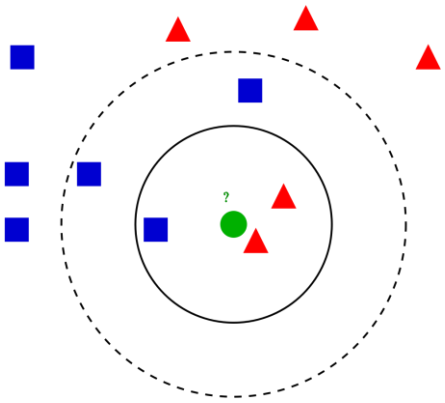
```
import random, time
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from pandas import read_csv
from sklearn.metrics import confusion_matrix, accuracy_score,
balanced_accuracy_score
from sklearn.preprocessing import StandardScaler

seed=random.seed(time.time())

filename = '../datasets/iris.data'
col_names=[  'sepal length', 'sepal width','petal length','petal
width','class']
data = read_csv(filename, names=col_names)
```

```
X = data[data.columns[:-1]]
Y = data['class']

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size =
0.5, random_state=seed)

stdScaler = StandardScaler().fit(X_train)
X_train = stdScaler.transform(X_train)
X_test = stdScaler.transform(X_test)

modelo = LogisticRegression( )
modelo.fit(X_train, y_train)
y_pred = modelo.predict(X_test)


cm =  confusion_matrix(y_test, y_pred)
acc=  accuracy_score(y_test, y_pred)
bacc = balanced_accuracy_score(y_test, y_pred)
print(cm)
print(acc)
print(bacc)
print(sklearn.metrics.cohen_kappa_score(y_test, y_pred))
```

6. K-VECINOS MÁS CERCANOS (KNN)

El modelo n-Nearest Neighbors (KNN) se trata de una técnica supervisada no paramétrica y no lineal de Machine Learning enormemente fácil de implementar que puede resolver problemas de clasificación y regresión bastante complejas. Debido a que no tiene una fase de entrenamiento como tal, es considerado un modelo *perezCálculoso*. Funciona de tal modo que para realizar una predicción utiliza todos los datos del conjunto de entrenamiento.

El principio que sustenta el modelo kNN es el más simples de todos los existentes en el grupo de los supervisados. Sin existir una fase de entrenamiento, simplemente se trata calcula la distancia de cada nuevo patrón de datos a todos los demás del conjunto de entrenamiento. El siguiente paso será seleccionar los k patrones más cercanos, donde k es un número entero. Finalmente, la predicción es el resultado de una combinación obtenida de la variable objetivo de los patrones seleccionados. Dicha combinación, en problemas de clasificación suele ser una mayoría, mientras que, en problemas de regresión, una media. La distancia para cuantificar la cercanía de dos patrones puede ser de cualquier tipo: euclídea, Manhattan, Minkowski... Por tanto, tan sólo se necesitan ajustar dos hiperparámetros para trabajar con kNN: el valor de k y la función de distancia. La **¡Error! No se encuentra el origen de la referencia.** muestra gráficamente cómo sería una predicción de un nuevo patrón, punto verde, mediante un modelo kNN, con dos valores diferentes de k.



El hecho de que sea un modelo perezoso, provoca que los modelos kNN sean extremadamente más rápidos que otros algoritmos que necesitan de una costosa fase entrenamiento. Por la misma razón, son un modelo que permite el aprendizaje on-line, es decir, que el conjunto de entrenamiento puede ser incrementado en cualquier momento con nuevas instancias.

Cabe mencionar que estos modelos suelen no tener un buen rendimiento ante conjuntos de datos con mucha dimensionalidad. Si el conjunto de datos

Ilustración 3: representación gráfica del proceso de predicción de un patrón con un modelo kNN (k=3 linea continua, k=5 línea discontinua).

cuenta con muchas instancias, el problema viene en la fase de predicción, en la que el coste computacional asociado a calcular las distancias con todo el conjunto es grande. Si el conjunto de datos cuenta con muchos atributos, el problema viene del coste computacional asociado a calcular cada distancia en sí.

7. Implementación en Python de un kNN

Por sus características, los modelos kNN son una técnica apropiada para resolver tanto problemas de clasificación binaria o multiclase, como problemas de regresión.

En Python es posible implementar este tipo de algoritmo utilizando el módulo `neighbors` de scikit-learn.

7.1. kNN para problemas de clasificación binaria

En Python es posible implementar un modelo kNN para resolver problemas de clasificación binaria utilizando la clase `KNeighborsClassifier` dentro del módulo `neighbors` de scikit-learn.

Los dos parámetros fundamentales con los que esta clase puede ser configurada son el valor de k con el parámetro `n_neighbors` (por defecto a 5), y la métrica de distancia a través del parámetro `metric` (por defecto 'minkowski').

Para ilustrar el siguiente ejemplo se hará uso del conjunto de datos *Indian-Liver-Patient* del repositorio UCI Machine Learning.

En todo caso, hay que recordar que cuando se trata con un modelo en scikit-learn, todas las variables deben ser de numéricas.

Por tanto, toda variable categórica en el conjunto de datos debe ser convertidas en numéricas del modo más adecuado y representativo posible.

```
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score
import pandas, random
from sklearn.preprocessing import RobustScaler
```

```
filename = '../datasets/Indian-Liver-Patient.csv'
col_names = ['age', 'gen', 'tbili', 'dbili', 'alkphos', 'sgpt',
'sgot', 'tp', 'alb', 'ag' ,'class']
data = read_csv(filename, names=col_names)

print(data.groupby('class').size())

#missing values
data.fillna(0, inplace=True)

X = data[data.columns[:-1]]
X = pandas.get_dummies(X, prefix=['gen'])
Y = data['class']

seed = random.randint(0,10)

X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=0.3, random_state=seed)

scaler = RobustScaler()
scaler.fit(X_train)
X_train  = scaler.transform(X_train)

model = KNeighborsClassifier(n_neighbors=7, metric='euclidean')
model.fit(X_train, Y_train)

X_test  = scaler.transform(X_test)
prediccion = model.predict(X_test)

print(confusion_matrix(Y_test, prediccion))
print(accuracy_score(Y_test, prediccion))
```

7.2. kNN para problemas de clasificación multiclase

La naturaleza de los modelos de kNN hace que estén diseñados de un modo directo para realizar problemas de clasificación multiclase. De este modo, no necesita de ninguna estrategia de combinación para afrontar este tipo de problemas.

Por tanto, en Python es posible implementar un modelo kNN para resolver problemas de clasificación multiclase utilizando igualmente la clase `KNeighborsClassifier` dentro del módulo `neighbors` de scikit-learn. Los parámetros fundamentales con los que esta clase puede ser configurada son los mismos que para clasificación binaria.

Para ilustrar el siguiente ejemplo se hará uso del conjunto de datos *Iris* del repositorio UCI Machine Learning.

En todo caso, hay que recordar que cuando se trata con un modelo en scikit-learn, todas las variables deben ser de numéricas. Por tanto, toda variable categórica en el conjunto de datos debe ser convertidas en numéricas del modo más adecuado y representativo posible.

```
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, balanced_accuracy_score
import pandas, random, time
from sklearn.preprocessing import StandardScaler
seed=random.seed(time.time())

filename = '../datasets/iris.data'
col_names=[  'sepal length', 'sepal width','petal length','petal
width','class']
data = read_csv(filename, names=col_names)

print(data.groupby('class').size())

X = data[data.columns[:-1]]
Y = data['class']

X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=0.3, random_state=seed)

scaler = StandardScaler()
scaler.fit(X_train)
X_train  = scaler.transform(X_train)

model = KNeighborsClassifier(n_neighbors=7, metric='euclidean')
model.fit(X_train, Y_train)

X_test  = scaler.transform(X_test)
prediccion = model.predict(X_test)

print(confusion_matrix(Y_test, prediccion))
print(balanced_accuracy_score(Y_test, prediccion))
```

7.3. kNN para problemas de regresión

En Python es posible implementar un modelo kNN para resolver problemas de regresión utilizando la clase `KNeighborsRegressor` dentro del módulo `neighbors` de scikit-learn.

Los dos parámetros fundamentales con los que esta clase puede ser configurada son el valor de k con el parámetro `n_neighbors` (por defecto a 5), y la métrica de distancia a través del parámetro `metric` (por defecto 'minkowski'). Para ilustrar el siguiente ejemplo se hará uso del conjunto de datos *housing* del repositorio UCI Machine Learning.

En todo caso, hay que recordar que cuando se trata con un modelo en scikit-learn, todas las variables deben ser de numéricas. Por tanto, toda variable categórica en el conjunto de datos debe ser convertidas en numéricas del modo más adecuado y representativo posible.

```
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error
import  random, time, numpy
from sklearn.preprocessing import StandardScaler
seed=random.seed(time.time())

filename = '../datasets/housing.csv'
col_names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS',
'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']
data = read_csv(filename, names=col_names, sep=" ")

X = data[data.columns[:-1]]
Y = data['MEDV']

X_train, X_test, y_train, y_test = train_test_split(X, Y,
test_size=0.3, random_state=seed)

scaler = StandardScaler()
scaler.fit(X_train)
X_train  = scaler.transform(X_train)

model = KNeighborsRegressor(n_neighbors=7, metric='euclidean')
model.fit(X_train, y_train)

X_test  = scaler.transform(X_test)
y_pred = model.predict(X_test)

rmse = (numpy.sqrt(mean_squared_error(y_test, y_pred)))
print(rmse)
```

8. PUNTOS CLAVE

En esta lección se ha aprendido:

- Entender los conceptos fundamentales de la Regresión Lineal simple y múltiple.
- Implementar en scikit-learn una Regresión Lineal para problemas de regresión.
- Comprender los conceptos fundamentales de la Regresión Logística simple y múltiple.
- Implementar en scikit-learn una Regresión Logística para problemas de clasificación binarios y multiclase.
- Exponer los conceptos fundamentales en los que están basados los modelos kNN.
- Implementar en scikit-learn diferentes modelos de kNN para problemas de clasificación binarios y multiclase y para problemas de regresión.